I. Le bloc PL-SQL

A. Introduction

Le PL/SQL est un langage de programmation développé aux débuts des années 1990s par Oracle, une entreprise américaine créée en 1977, dont l'un des principaux produits est le célèbre SGBD *Oracle Database*.

Le PL/SQL est un langage **procédural**, qui constitue une extension du langage SQL, qui lui, ne sert qu'à la manipulation des données.

Cela donne au PL/SQL un avantage énorme : pouvoir mélanger dans un même traitement :

- La puissance des instructions de manipulation de données du langage SQL *(requêtes de sélection, insertion, modification, …)*
- Avec la souplesse et les techniques procédurales que possède un langage de programmation comme C ou Java *(utilisation de variables, de structures alternatives & itératives, de fonctions, la gestion des exceptions, …)*

Et ceci avec pratiquement la même syntaxe.

Ces traitements peuvent être exécutés, soit directement dans de simples blocs PL/SQL, soit à partir de certains « objets de base de données » comme les procédures stockées et les packages.

S.ELYAHYAOUI - PLSQL

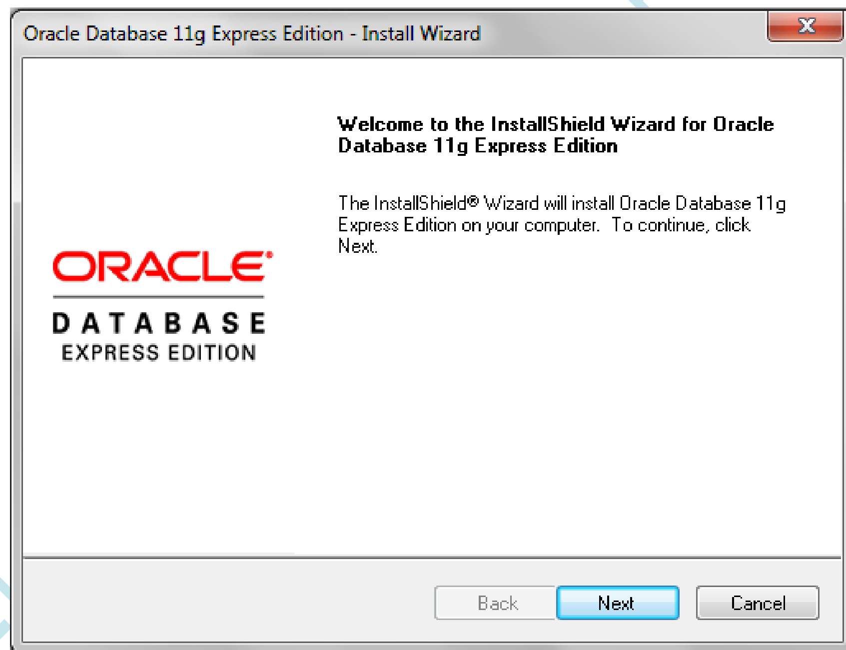
B. Environnement de travail

1. Oracle version 11g

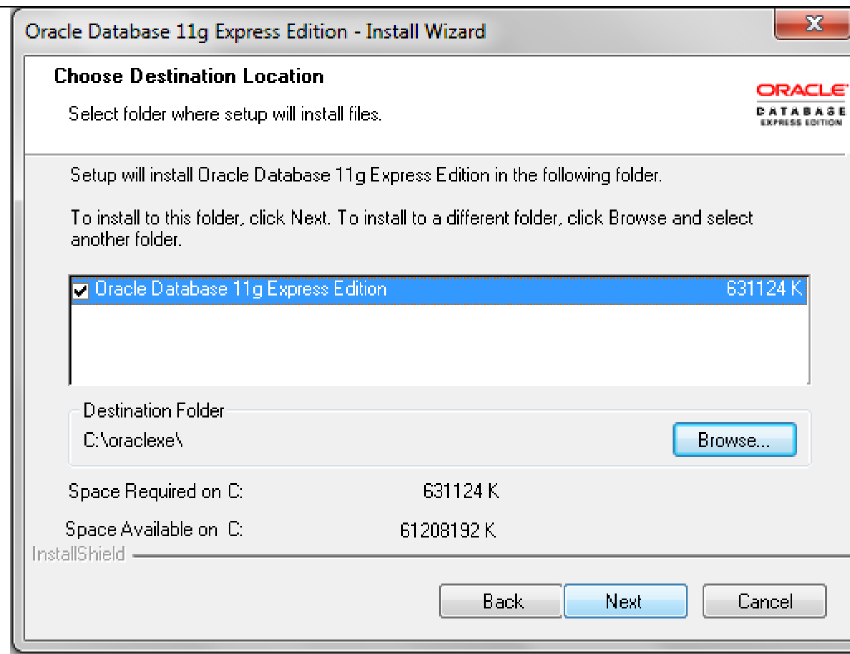
Le SGBD Oracle est un logiciel payant, ce tutoriel s'appuiera donc sur l'édition **Express Edition** (**XE**) de la version d'**Oracle 11g** (*la dernière version est **Oracle 12c***), qui est une version limitée mais téléchargeable gratuitement sur le site d'Oracle :

http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html

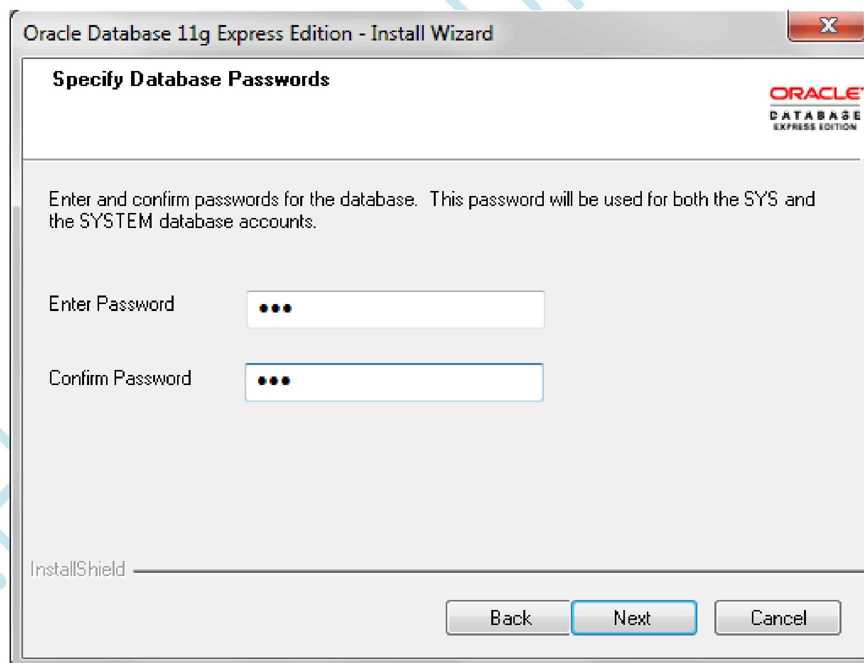
Limitations de cette version :
- Aspects de performances et de sécurité réduits.
- Espace dédié au stockage de la base de données limité à 11GO.
- Utilisation de la RAM limitée à 1GO.

Installation d'Oracle 11g XE
1- Lancer le programme d'installation qui se trouve dans le dossier disk1 
2- Validation

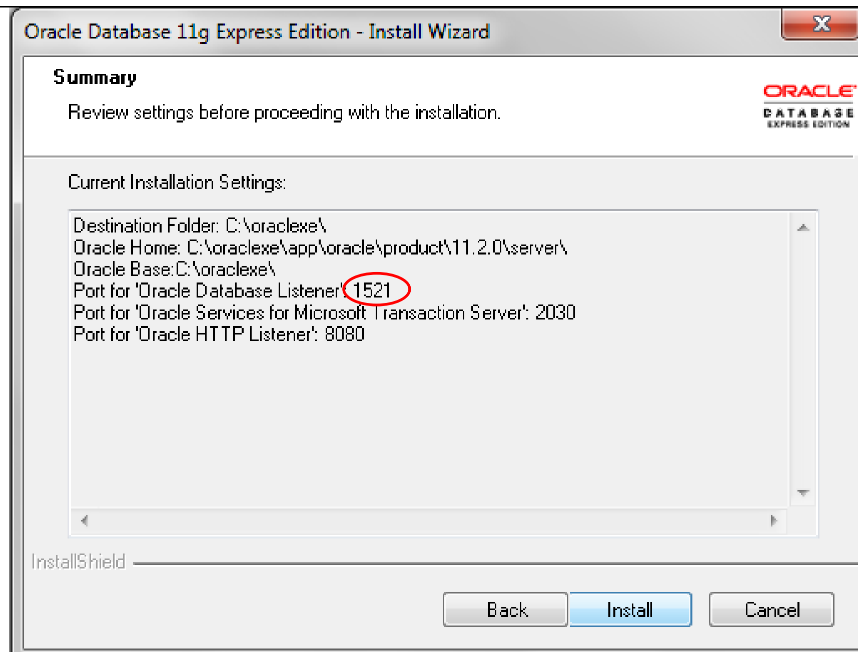
S.EL PL/SQL



3- Définir un mot de passe pour l'utilisateur **SYSTEM**, utilisateur qui sera créé et configuré automatiquement lors de l'installation de la version XE.
Le compte de cet utilisateur sera utilisé pour se connecter à la base de données.

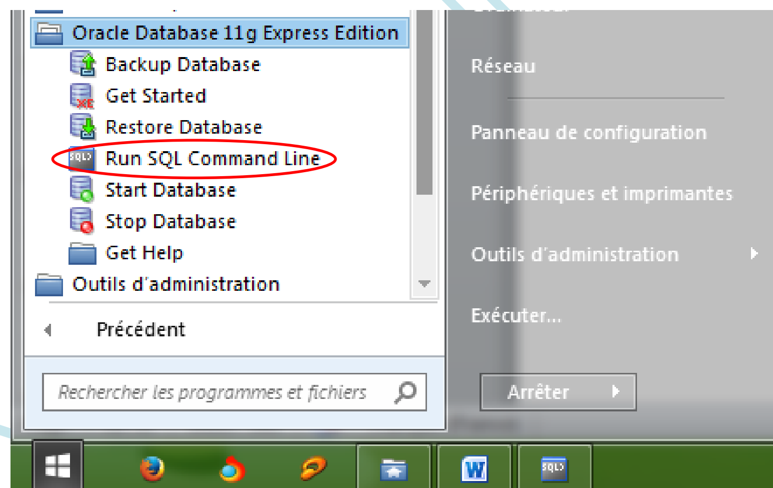


4- Noter la valeur du port de connexion à la base de données Oracle : **1521** par défaut

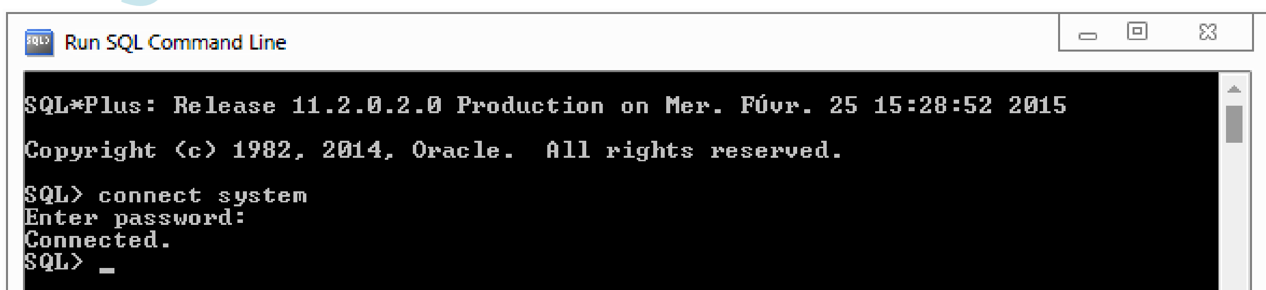


1ier test de connexion

1- Ouvrir l'invite de commandes SQL d'Oracle : ***SQL command line***.



2- Exécuter la commande SQL de connexion avec le compte d'utilisateur SYSTEM : " **connect system** ", taper ensuite le mot de passe défini lors de l'étape 3 de l'installation.



3- Connexion réussie.

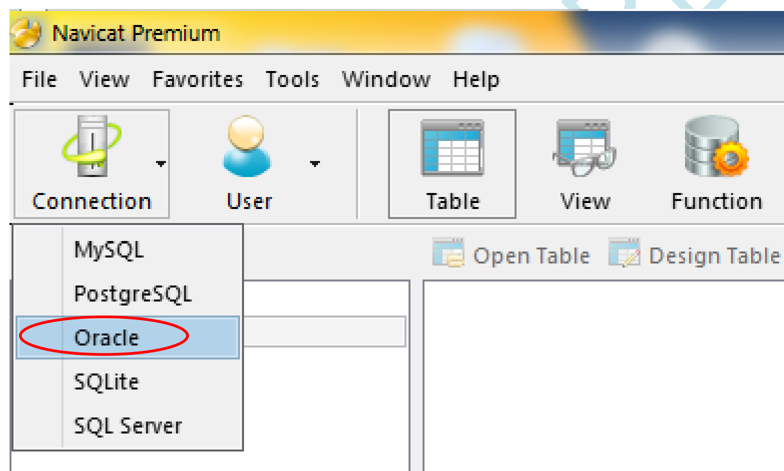
2. Navicat version 10

Bien sûr, il n'est pas très pratique de programmer, tester et exécuter tous ses codes PL/SQL avec le **SQL command line**. L'outil graphique prévu à cet effet et qui s'installe avec les versions complètes s'appelle ***SQL*Plus***, mais il n'est pas incorporé dans la version XE.
L'édition XE ne contient qu'une base de données installée et préconfigurée lors de l'installation du logiciel, ainsi que des services d'Oracle.

Evidemment, il existe d'autres logiciels pour se connecter à Oracle, et exécuter du code PL/SQL, comme ***SQLDeveloper*** (*un produit d'Oracle, téléchargeable gratuitement*) et ***Navicat***, un logiciel payant, sur lequel se basera ce tutoriel.
La version ***Premium*** de **Navicat** prend en charge plusieurs types de SGBD.

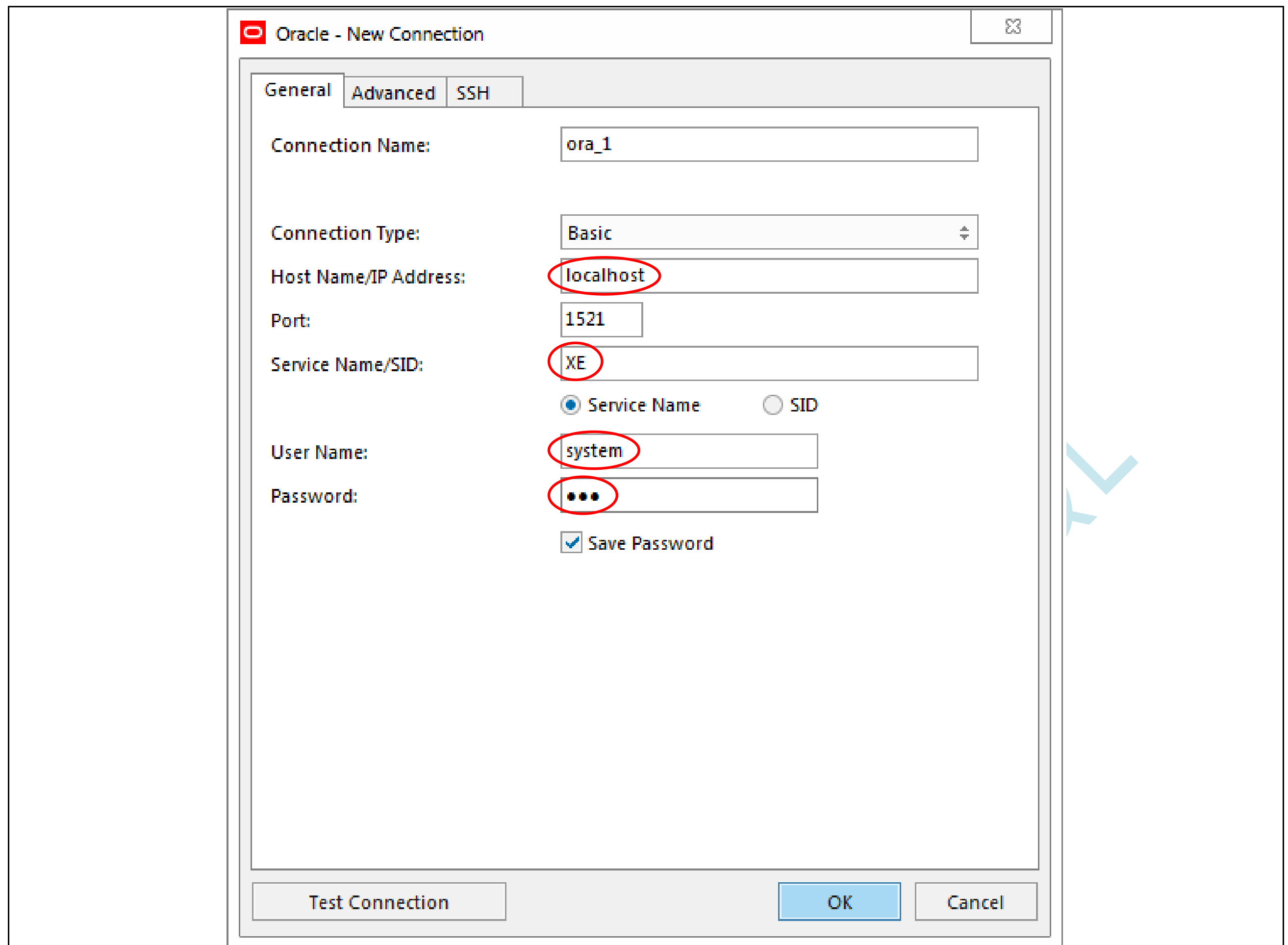
Se connecter à Oracle à partir de Navicat

1- Ouvrir Navicat, puis dans la rubrique « Connexion », choisir le type de base de données « Oracle ».

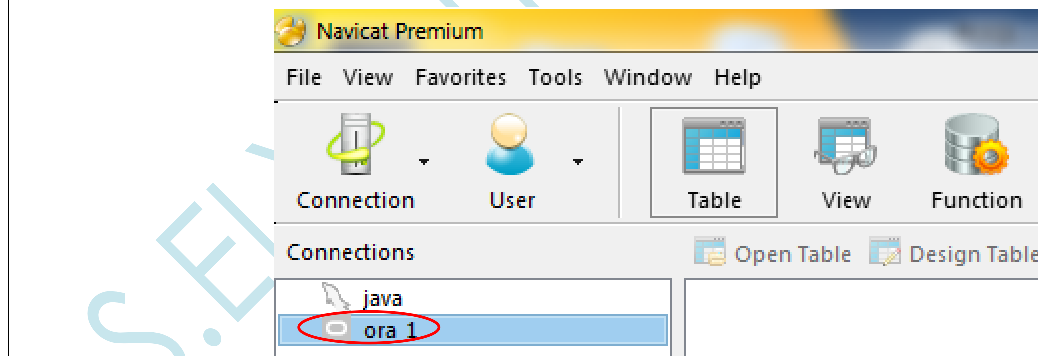


2- Dans la boite de dialogue qui s'ouvre, spécifier les paramètres de connexion :

- **Nom de la connexion**
- **Host Name / IP Adress** : Adresse IP de la machine distante sur laquelle est installé Oracle, sinon ***localhost*** s'il est installé sur la même machine que Navicat.
- Service Name : Nom du service Oracle (***XE***)
- User Name : ***system***
- Password : Le même mot de passe utilisé lors de la connexion avec **SQL command line**.
- Les autres paramètres sont renseignés par défaut.

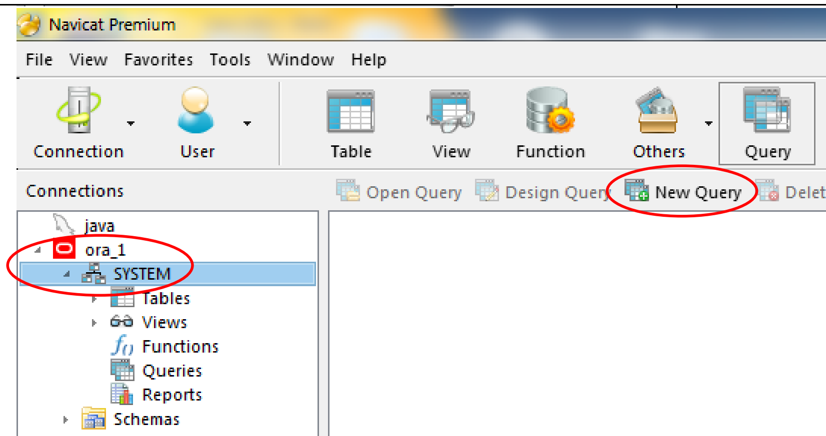


3- Tester la connexion, puis cliquer **OK** pour l'enregistrer dans la liste des connexions.



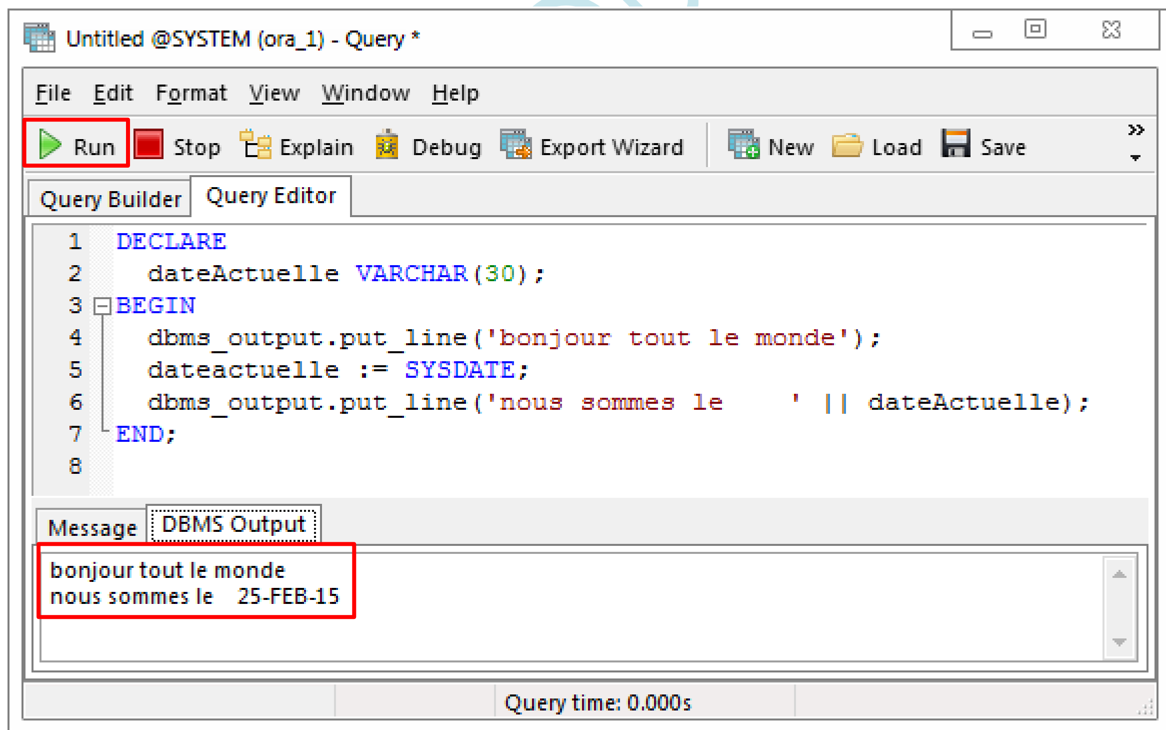
3. Exécuter un 1ier programme PL/SQL

Ecrire & exécuter un programme PL/SQL avec Navicat
1- Ouvrir la connexion récemment créée, puis ouvrir le compte d'utilisateur **SYSTEM**, puis dans la rubrique **Query** choisir l'action **New query**



2- Dans la fenêtre qui s'ouvre, taper & exécuter le code suivant (en cliquant sur le bouton **Run**), puis visualiser le résultat d'exécution dans l'onglet de la sortie d'affichage « **DBMS Output** » :

```
DECLARE
      dateActuelle VARCHAR(30);
BEGIN
      dbms_output.put_line('bonjour tout le monde');
      dateactuelle := SYSDATE;
      dbms_output.put_line('nous sommes le     ' || dateActuelle);
END;
```



C. Syntaxe de base

1. Structure d'un programme PL/SQL

Structure générale d'un programme PL/SQL

La syntaxe générale d'un bloc PL/SQL est comme ceci :

```
DECLARE  [optionnel]

BEGIN

EXCEPTION  [optionnel]

END;
```



IMPORTANT

- Le bloc **END** doit être suivi d'un **;**
- Chaque instruction doit être suivie d'un **;**
- Le non-respect de ces deux conditions donne une erreur d'exécution.
- On peut rajouter le caractère / pour marquer la fin du programme.

Exemple 1 : Le programme vide

```
BEGIN
     NULL;
END;
```


REMARQUES

- *Ce programme ne produit aucun effet lors de son exécution.*
- *L'instruction `NULL;` est appelée « l'instruction vide », elle n'a comme effet que l'action de passer à l'instruction qui la suit.*

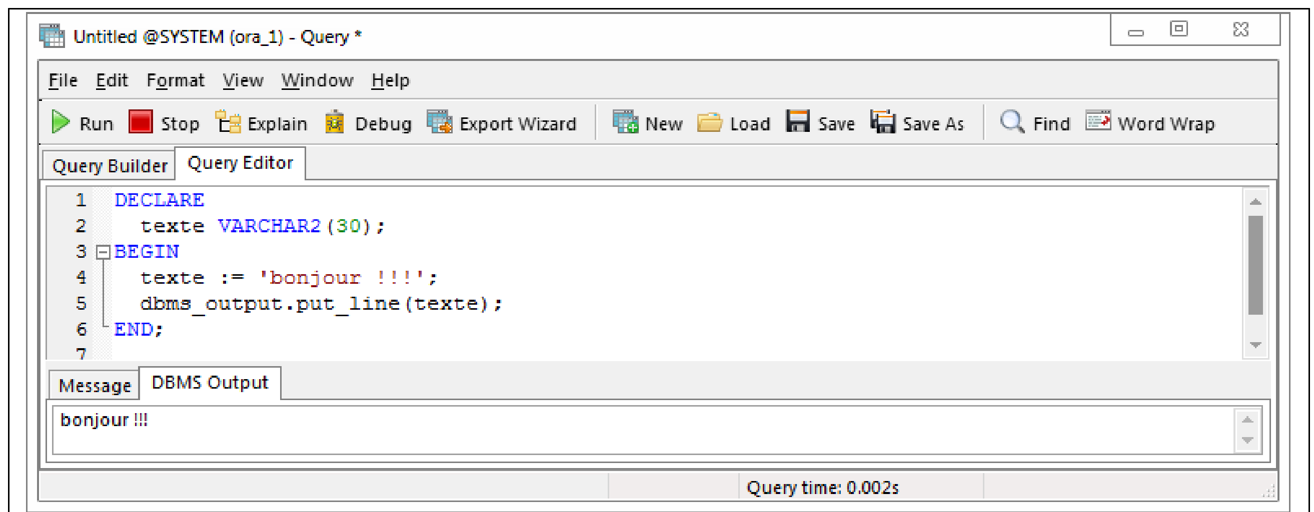
Exemple 2

```
DECLARE
     texte VARCHAR2(30);
BEGIN
     texte := 'bonjour !!!';
     dbms_output.put_line(texte);
END;
```



REMARQUE

La déclaration de variables et l'affichage de données seront détaillés dans les prochains paragraphes.



2. Les commentaires

Définition

Comme dans tous les langages de programmation, les commentaires ne sont pas pris en compte dans l'exécution d'un programme PL/SQL.

Deux types de commentaires :
Commentaires mono-ligne : **--**

```
-- ceci est un commentaire
```

Commentaires multi-ligne : **/* */**

```
/*
   ceci
   est un
   commentaire
*/
```


IMPORTANT

Un programme ne doit pas se terminé par un commentaire multi-ligne, sinon on doit rajouter le caractère de fin /

Exemple

✔	✘
DECLARE texte VARCHAR2(30); BEGIN texte := 'bonjour !!!'; dbms_output.put_line(texte); END; /* Un commentaire *//	DECLARE texte VARCHAR2(30); BEGIN texte := 'bonjour !!!'; dbms_output.put_line(texte); END; /* Un commentaire */

S.ELYAHYAOUI - PLSQL

3. Les étiquettes et les sauts d'instructions

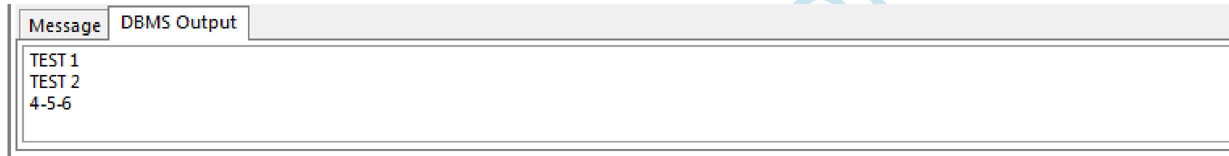
Définition

Une étiquette ou label, est un nom, inséré entre les délimiteurs **<<** et **>>**, pour désigner une partie du code.
On peut sauter vers cette partie en utilisant l'instruction :
GOTO nom_de_l'étiquette;

Exemple

```
BEGIN
      dbms_output.put_line('TEST 1');
      GOTO une_etiquette;
      dbms_output.put_line('1-2-3');

      <<une_etiquette>>
      dbms_output.put_line('TEST 2');
      dbms_output.put_line('4-5-6');
END;
```



 **IMPORTANT**

Une étiquette doit contenir au moins l'instruction vide.
On obtient une erreur d'exécution si l'étiquette ne contient rien.

✔	✖
`BEGIN` `GOTO une_etiquette;` `<<une_etiquette>>` `NULL;` `END;`	`BEGIN` `GOTO une_etiquette;` `<<une_etiquette>>` `END;`

S.ELYAHYAOUI PL/SQL

D. Les principaux types de données

1. Le type numérique

Définition : NUMBER(a,b)
Nombre réel avec ***a*** le nombre total de chiffres, et ***b*** le nombre de chiffres à droite de la virgule.
Exemples
a NUMBER(4); **b NUMBER;** **c NUMBER(5,2);**

2. Les types « chaine de caractères »

Définition : CHAR(n)
Chaîne de caractères de longueur **fixe n**, avec **n** compris entre **1** et **32767**. Si aucune taille maximale n'est précisée alors la valeur utilisée par défaut est 1.  IMPORTANT La capacité d'une variable PL/SQL de type **CHAR** est beaucoup plus supérieure à celle d'une colonne de table de même type, qui ne supporte que 2000 caractères au maximum.
Définition : VARCHAR2
Idem que CHAR, sauf que la longueur est **variable**.  IMPORTANT La capacité d'une variable PL/SQL de type **CHAR** est beaucoup plus supérieure à celle d'une colonne de table de même type, qui ne supporte que 4000 caractères au maximum.

3. Les sous-types

Sous-types de NUMBER
DEC, DECIMAL, NUMERIC, DOUBLE PRECISION, FLOAT, REAL INTEGER, INT, SMALLINT, …
Sous-types de VARCHAR2
VARCHAR, NCHAR, NVARCHAR, …

4. Autres types de données

Le type « boolean »
Ce type de données n'accepte que **3** valeurs possibles : **true**, **false** ou **NULL**. Ce type est utilisé comme résultat d'expressions logiques, mais on ne peut l'utiliser ni dans des instructions, ni dans des fonctions.
Exemple 1

```
DECLARE
      b boolean;
BEGIN
      b := 1 > 2;
      IF b THEN
            dbms_output.put_line('test 1');
      END IF;
      dbms_output.put_line('test 2');
END;
```

```
Message | DBMS Output
test 2
```

Exemple 2

```
DECLARE
      b boolean;
BEGIN
      b := true;
      IF b THEN
            dbms_output.put_line('test 1');
      END IF;
      dbms_output.put_line('test 2');
END;
```

```
Message | DBMS Output
test 1
test 2
```

Le type « DATE »

Stocke une date dans un format de longueur fixe, ayant l'aspect **'DD-MON-YY'** ou **'DD-MON-YYYY'**

Le type « TIMESTAMP »

Idem que le type **DATE**, mais en ajoutant les infos heure, minute, secondes et fractions de secondes.

Exemple

```
DECLARE
      d1 DATE;
      d2 TIMESTAMP;
BEGIN
      d1 := '23-JAN-15';
      d2 := '23-JAN-15';
      dbms_output.put_line(d1);
      dbms_output.put_line(d2);
END;
```

```
Message | DBMS Output
23-JAN-15
23-JAN-15 12.00.00.000000 AM
```

Les types définis par l'utilisateur

Comme le langage C par ex. (***typedef***), le PL/SQL permet de définir des types dont les noms seront choisis par le programmeur.
Ces nouveaux types seront définis sur la base des types prédéfinis dans PL/SQL.

`SUBTYPE nom_du_sous_type IS type;`


IMPORTANT

La définition d'un nouveau type **ne signifie pas la création d'une variable**.

Exemple

```
DECLARE
      SUBTYPE entier IS INTEGER;
BEGIN
      NULL;
END;
```

Création d'un nouveau type appelé « ***entier*** »

S.ELYAHYAOUI - PLSQL

E. Déclaration & affectation de variables & constantes

1. Déclaration & initialisation d'une variable

Syntaxe : Déclaration

```
DECLARE
      nom_variable TYPE;
```

 **REMARQUE**

Comme SQL, le PL/SQL est insensible à la casse.

 **IMPORTANT**

Le nom d'une variable :

- Ne doit pas être un mot-clef réservé de PL/SQL.
- Peut être composé de lettres, de chiffres et des caractères _ **$** et **#**
- Ne doit pas débuter par les caractères _ **$** ou **#**

Exemple

```
DECLARE
      a NUMBER;
      b NUMBER(3,1);
      SUBTYPE entier IS INTEGER;
      i entier;
BEGIN
      b := 22.5;
END;
```

:= opérateur d'affectation

Exemple : Initialisation lors de la déclaration

```
DECLARE
      a NUMBER;
      b NUMBER(3,1);
      SUBTYPE entier IS INTEGER;
      i entier := 10;
BEGIN
      b := 22.5;
END;
```

La clause NOT NULL

Syntaxe :

```
            nom_variable TYPE NOT NULL := valeur_par_defaut;
```

Utilisée pour empêcher l'affectation de la valeur ***NULL*** à la variable, mais dans ce cas La valeur par défaut est obligatoire.

Exemple

```
DECLARE
      d1 VARCHAR2(20) NOT NULL := 'texte';
BEGIN
      dbms_output.put_line(d1);
      d1 := 'un autre texte';
      dbms_output.put_line(d1);
END;
```

S.ELYAHYAOUI - PLSQL

Message | DBMS Output

```
texte
un autre texte
```

Déclaration à partir d'un type dérivé

On peut aussi définir une variable à partir du type de l'une des colonnes d'une table existante.
Syntaxe :

```
nom_variable   "nom_table"."nom_colonne"%TYPE;
```

Exemple

```
DECLARE
      d1 "une_table_de_test"."name"%TYPE;
BEGIN
      d1 := 'ABC';
END;
```

2. Les constantes

Définition : Constante

Une constante est une variable **qui doit être initialisée lors de la déclaration**, et dont l'affectation dans le bloc ***BEGIN*** donne **une erreur d'exécution**.
Syntaxe :

```
nom_variable constant TYPE := valeur;
```

Exemple

```
DECLARE
      d1 constant VARCHAR2(20):= 'texte';
      d2 constant NUMBER;        <-- Erreur

BEGIN
      d1 := 'texte';             <-- Erreur
      dbms_output.put_line(d1);
END;
```

S.ELYAHYAOUI - PL SQL